National Research Council Canada
Conseil national de recherches Canada

Division of Electrical Engineering
Division de génie électrique

ERB-1002

ALTER-Harmony: Control of a PUMA Robot from the Chorus Multiprocessor

D.H. O'Hara, S. Elgazzar, and G. The

December 1987

Canada

NRCC No. 28492

88-04449/

Copyright 1987 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1987 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 301
Division of Electrical Engineering
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R8

Des exemplaires supplémentaires gratuits peuvent être obtenus à l'adresse suivante:

Bureau des publications, Pièce 301
Division de génie électrique
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R8

ALTER-HARMONY: CONTROL OF A PUMA ROBOT FROM THE CHORUS MULTIPROCESSOR

D.H. O'Hara, S. Elgazzar, and G. The

ERB-1002
NRCC No. 28492

DECEMBER 1987

TABLE OF CONTENTS

	Page
ABSTRACT/RÉSUMÉ	v
INTRODUCTION	1
UNIMATION PERSPECTIVE VS. COMPUTING TECHNOLOGY IMPLEMENTATION OF ALTER	1
ALTER FUNCTIONALITY	
START_ALTER	5
STOP_ALTER	5
IDLE	6
DEFINE_FIXED_FRAME	6
MOVE	6
DETAILED DESCRIPTION OF ALTER FUNCTIONALITY	6
START_ALTER	7
STOP_ALTER	7
IDLE (ec)	8
DEFINE_FIXED_FRAME	8
MOVE	8
ANGLE DEFINITION	9
Methods for Describing Position and Orientation	9
Relationship Between the Different Representations	12
SUMMARY	17
ACKNOWLEDGEMENT	17
REFERENCES	17
APPENDIX A	19

FIGURES

1	World and tool coordinate frames	3
2	Position and orientation vectors at end effector	10
3	(a) Euler angles; (b) RPY angles	11

TABLE

1	Content of normal messages from VAL II	2

ABSTRACT

The PUMA family of robots may be controlled through an external channel for realtime path modification. This link is called ALTER control. As conceived by the manufacturer, this link transmits data to the controller to modify a preprogrammed path. As described here ALTER is used to specify the total motion of the robot. A capability is provided for Cartesian motions in workspace coordinates, i.e., full control of position and orientation. Two methods of specifying orientation are used. Angle specification in VAL II programs uses the Euler angle nomenclature whereas ALTER uses the roll-pitch-yaw description. The relationship between these two and their relationship to the homogeneous transformation for describing position and orientation are described.

RÉSUMÉ

Les robots de la famille PUMA peuvent être commandés par l'intermédiaire d'un canal extérieur dans le but de modifier les trajets en temps réel. Cette liaison est appelée "commande ALTER". Le fabricant l'a conçue de façon qu'elle transmette des données au contrôleur dans le but de modifier un trajet préprogrammé. Le présent rapport décrit comment la liaison ALTER est utilisée pour spécifier le mouvement complet du robot. Un dispositif assure des mouvements cartésiens dans les coordonnées de l'espace de travail, c'est-à-dire une commande totale de la position et de l'orientation. L'orientation est spécifiée de deux façons. Les programmes VAL II utilisent la nomenclature d'Euler pour spécifier les angles, tandis que la liaison ALTER utilise le roulis, le tangage et le lacet. La relation entre les deux méthodes et leur relation avec la transformation homogène utilisée pour décrire la position et l'orientation sont décrites.

ALTER-HARMONY: CONTROL OF A PUMA ROBOT FROM THE CHORUS MULTIPROCESSOR

D.H. O'Hara, S. Elgazzar, and G. The

INTRODUCTION

This report is written for the user of PUMA 560 robot in the robotics laboratory of the Computing Technology Section. These facilities have been developed to perform experiments in sensor-based robotics. The aim here is the flexible position control of the PUMA robot, in one of four coordinate frames: (a) robot world coordinate frame, (b) a frame fixed to the tool frame at the beginning of a motion sequence, (c) an arbitrary frame fixed in space, (d) dynamic tool frame. The distinction between these is that frames (a), (b), and (c) are fixed frames and frame (d) is a dynamic frame.

This report plus the Unimation Programming Manual - Part 3 Real-Time Path Control will provide sufficient understanding of the facility for its use. The report describes the communication with the Unimation controller from an external computer, the low level functions that have been written to support this communication, and the functions available to a user from the Harmony* operating system used in the laboratory. Harmony and Chorus, the multiprocessor upon which it runs, are described in references 1 and 2. The approach here differs from that of Unimation, first, to accommodate our special requirements and, second, to overcome anomalies in the Unimation software. The report also provides a comprehensive description of tool orientation. Angle specification in the VAL II language uses the Euler angle nomenclature whereas the ALTER format requires the roll-pitch-yaw description. The conversion between these two and their relationship to the homogeneous transformation for describing position and orientation are explained.

UNIMATION PERSPECTIVE VS. COMPUTING TECHNOLOGY IMPLEMENTATION OF ALTER

ALTER is the VAL II provision for providing realtime path control. By this is meant the ability to modify a program-directed position and orientation (pose) of the robot tool while the robot is moving. Data for real-time path modification may be obtained from two sources: the external computer over a serial link or a user-written VAL II program. These two situations are referred to as "external" and "internal" ALTER. ALTER provides two-way communication with the external computer, giving useful

---

* Mark reserved for the exclusive use of Her Majesty the Queen in right of Canada by Canadian Patents and Development Ltd.

information on the state of robot motion. ALTER also provides a total of four straight line motion options in world and tool coordinates in either cumulative or noncumulative modes. In the cumulative mode, the effect of any data received is accumulated and the robot location is modified by the sum of all past ALTER data. In the noncumulative mode, the robot location is modified only by the most recent data. The complete capability of the ALTER provision is described in the VAL II manual (3). As discussed, ALTER permits robot motion to be controlled from an external computer, but a variety of information from the controller is available to the external computer via the ALTER link also. Most important of this controller information is the transformation data describing the position and orientation of the tool frame. The details are provided in the VAL II manual (3). Table 1 summarizes this controller information.

In the Computing Technology Section implementation, ALTER is used only with an external computer and in the world cumulative mode. Path

Table 1. Content of normal messages from VAL II

Byte	Name	Description
1	Control byte	Control information to the external computer.
2	Segment state	Current motion segment state such as "Motion Stopped" (i.e., information related to the robot motion).
3-4	Segment number	Set to 1 for the first motion segment after an ALTER instruction is executed; incremented by 1 for each new segment (i.e., each new motion instruction).
5-6	Motion percent	The percentage completion of the current segment. Starts at 0%; stops at 100% if the path is broken. Scaled so that 100% is represented as 16384 (40000 octal).
7-10	Transformation	Optional transformation data as selected by the ALTER mode parameter. These data represent either the current setpoint or the actual location read from the robot. If transformation output is not selected in the ALTER instruction, bytes 7-30 are not transmitted.

modification can be applied to straight line motion and to the DELAY instruction only. If the DELAY instruction is interpreted as a move to "nowhere," then total motion of the arm is specified by the path modification data supplied from the external computer. We use the DELAY instruction only. The VAL II controller imposes two significant constraints: a 28 ms cycle time and a communication response requirement. The Unimation controller must initiate the ALTER communication and treat the external computer as a peripheral which must respond to its stimuli or an error will occur and communication will terminate. This philosophy requires that we have another link to the VAL controller to execute a VAL program whose principal function is to initiate communication over the ALTER link to the external computer. This second link is provided using a terminal emulator mode. The software to do this is called VALPAK (4).

ALTER as described by Unimation allows robot control in two coordinate frames and in two modes in each frame. Unimation refers to these as world coordinates; the two modes are cumulative and noncumulative for a total of four operating states. These two frames are shown in Fig. 1. The tool

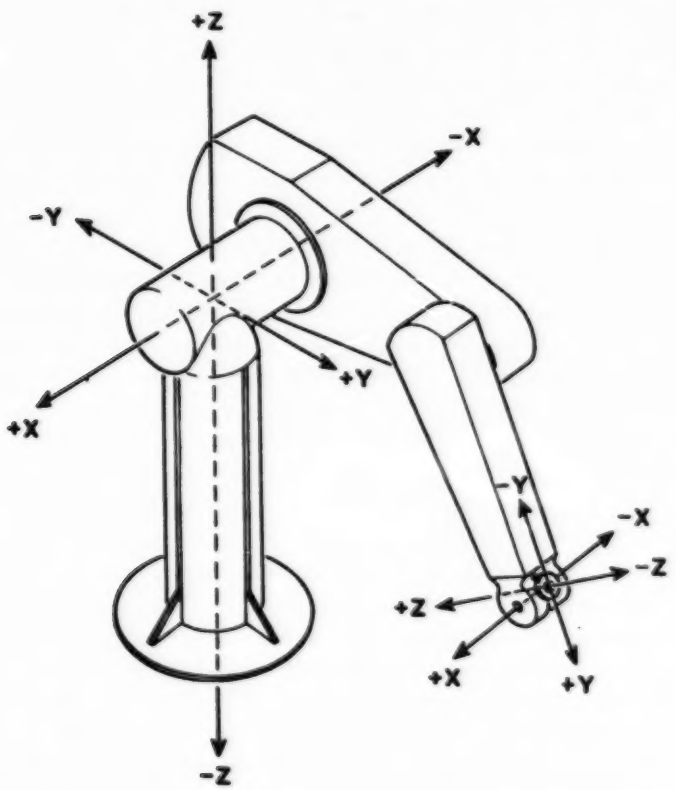

Figure 1. World and tool coordinate frames.

frame shown here is the null tool frame. In general the origin of the tool frame will be located on the end of arm tooling; it is the pose of this point which is controlled by motion commands. The ALTER data can modify all six degrees of freedom of the tool.

In the laboratory tests with ALTER we were unable to achieve these four modes. The problems encountered were of the type of inconsistent motions within a command type and accumulated errors which coupled changes into parameters where none had been specified. The world cumulative state, however, performed predictably and reliably. Building on this one mode, we have implemented complete robot motion control with reference to four coordinate frames.

The four frames in which motions may be specified are (a) robot world frame, (b) a fixed frame at the current tool frame, (c) a fixed frame arbitrarily specified, and (d) a dynamic tool frame. These four coordinate frames are a sufficient description for sensor-based position control. Typical applications of each of these frames are outlined below. The world coordinate frame is the reference frame and in fact is the frame to which we translate all motion commands before sending them to the controller over the ALTER link. It is generally located at the PUMA shoulder as shown in Fig. 1 but may be shifted through the VAL II BASE command. The fixed frame at the current tool frame mode (b) is specified to be that at the beginning of the current ALTER sequence and remains fixed at that location until the ALTER sequence terminates. The frame representation would be used in a situation where data are being collected from a wrist-mounted camera for the purpose of planning a set of movements based on that image. Subsequent evaluations would be made at the end of each motion sequence. The fixed frame (c) is a generalization of the fixed tool frame. It may be defined interactively by moving the tool tip to the desired pose or may be specified directly in the format of the **T** matrix (to be described later). A fixed overhead camera would reference such a frame. The dynamic tool frame is again a familiar representation to VAL users. It is the frame in which the teach pendant "TOOL" operates. In Fig. 1, the null tool is shown. However, any physical tool added to the wrist may be defined to the controller and motions then are referenced to the frame so defined. As the tool moves so does the origin of the tool frame and, therefore, the description dynamic. Motion control requiring immediate feedback will use this representation. Insertion of the prismatic block using the wrist-mounted six degree of freedom displacement sensor (Draper Labs IRCC) (5) is typical. The error displacements are measured directly in the tool frame and corrective motions are specified in this moving frame. The insertion demonstration described in ref. 5 was the first application of the ALTER support software described here.

The robot motions are specified to the VAL II controller in the world coordinate frame. Motions in the other frames must be converted to world coordinates. This is done following the procedure outlined in Paul "Transforming differential changes between coordinate frames," ref. 6, sect. 4.4,). As we are considering only incremental motions, we can consider all motions relative to the tool location. Furthermore, in describing rotations in world coordinates we visualize rotation in a frame with origin at the tool but with axes parallel to the world axes and not about the origin of the true world axes at the PUMA shoulder. Consequently, unlike Paul (6), we do not have to consider the displacement from the origin of the world frame.

The expression used is $\Delta^W = C\Delta^S$ where Δ^W represents increments in world coordinates, Δ^S are increments in one of the secondary frames, and **C** is the conversion matrix relating the two. The **C** matrix is just the rotation portion of the **T** matrix describing the location of the robot tip, or again following Paul's notation [n, o, a]. When describing motions in tool coordinates, this is particularly convenient since **C** transformation values are available on each ALTER communication cycle from VAL II.

ALTER FUNCTIONALITY

START_ALTER

In order for ALTER to begin, both the PUMA controller and the external computer must be ready to exchange data. Since the PUMA controller controls the communication, the external computer must be waiting to respond when the controller initiates transmission. There are a number of ways of satisfying this dependency; the one we have chosen is to have the initializing routines for the external computer run in the same processor and at a higher priority than the one which initiates the controller communication. Once the external computer task has run to a point where it is ready to begin communication with the controller, it blocks and only then is the task which initiates the VAL II program "DEMOALT" able to run. This function is called once for each motion sequence in ALTER.

STOP_ALTER

This function stops ALTER. All communication over the ALTER link is terminated. Again the PUMA controller actually terminates the process. ALTER provides a limited data transfer from the external computer in addition to displacement data. The mechanism for this transfer is the exception code, a number between 0 and 127 which is then available to a VAL II

program. We use the number 127, as STPCOD, to cause such a program to execute the VAL II command NOALTER and terminate ALTER.

IDLE

This function maintains the robot in its current location. The exception code is also transmitted.

DEFINE_FIXED_FRAME

As described earlier, all motions are specified to the controller in the world coordinate frame of the robot. If we wish to specify motion in any other frame, then we must supply the transformation relating the new frame to the world frame, the transformation **C**, previously described. Again, since we are specifying relative motions we need only the rotation portion of the full transformation matrix. The user must supply the elements of the rotation matrix scaled by 16384 and in column order. These values are supplied in integer in the array FRAME_BUFF. In the current implementation, an IDLE command is also sent to satisfy the 28 ms cycle.

MOVE

This function moves the the robot in six degrees of freedom in Cartesian space. An exception code is also transmitted. Before this function is called, the variable FRAME must be set to 0 for World, 1 for Tool, or 2 for Fixed. Linear displacements are in millimetres and rotations are in degrees. Displacement data are stored in the array MOTION_BUFF[6] in floating point notation as follows:

MOTION_BUFF[1]	translation in X
MOTION_BUFF[2]	translation in Y
MOTION_BUFF[3]	translation in Z
MOTION_BUFF[4]	rotation about X
MOTION_BUFF[5]	rotation about Y
MOTION_BUFF[6]	rotation about Z

DETAILED DESCRIPTION OF ALTER FUNCTIONALITY

This will elaborate on the functionality of ALTER as provided by the task Alt_server(). The functions are START_ALTER, STOP_ALTER, IDLE, MOVE, and DEFINE_FIXED FRAME. One of these functions must be active once each 28 ms. The application program must respond with control and data at this rate. The actual synchronization is provided by the message passing mechanism triggered by the ALTER communication data transmission. All the programs discussed in this section are listed in Appendix A.

START_ALTER

This description of the starting sequence relates to the code for Alt_server(). The function Ini_alt_serv() initializes the routine _Vavpint which is the "handler" for the 19.2 kbaud serial line to the VAL II controller. It also creates the tasks _In_packet and _Out_packet which, as their names indicate, coordinate messages to and from the Val II controller. For the case START_ALTER, function Ini_Alter() sets the global variable _Out_packet = 0 and then blocks waiting for _In_packet to send to it. The variable _Out_packet informs the routine _Vavpint that this is to be an initial communication transfer. As Alt_server() is now blocked, the task which starts the communication from the VAL II controller via the program, DEMOALT, can now run.

The first message from VAL II unblocks Alt_server() which then through Ini_Alter() unblocks _Out_packet by receiving and replying to it. The Ini_Alter() function then returns, allowing the START_ALTER case to run to completion. ALTER protocol requires a response to this first VAL II message with the special acknowledge message. This is satisfied by _Out_packet. The next message from VAL II will be a normal message and will be treated as such by _In_packet, passing it through the Alt_server() task to the application task. However, _Out_packet treats the first normal message as a special case again. The Vavpint routine must begin responding to the incoming message from VAL II after the second character to meet the communication timing requirements. Normally, the application program would have generated an appropriate command and this would have been passed to _Out_packet, but, since all outgoing messages are one communication cycle behind, none exists for this cycle. This is compensated for by an "idle" or "move no where" return message to the VAL II controller provided by the Vavpint: Out_packet task. The communication initialization is now complete and the in/out message sequence between the external computer and the VAL II controller must be maintained on a 28 ms cycle.

STOP_ALTER

The application program will terminate ALTER by selecting the case STOP_ALTER in Alt_server(). If ALTER is running, i.e., ALTER = UP, then the function Stp_Alter() is called. A stop ALTER command, STPCOD, will be sent to the VAL II controller if there is no communication error. Regardless of whether or not there is an error, this function tries to maintain communication with the controller and sends "no motion" commands until a VAL "STPMSG" is received, indicating VAL II is terminating ALTER communication.

The first byte in the normal ALTER message from the external computer is the exception code. We have set STPCOD equal to 127. This code is

retrieved by a VAL II process control program RDCODE using the VAL II function Exception. When it is detected, the communication loop in DEMOALT is ended and a VAL II NOALTER command is executed, terminating communication with the external computer.

IDLE (ec)

The idle command is used when it is desirable to maintain communication with the controller but we do not want to actually move the manipulator. The variable "ec" may be set to a value in the range 0 to 126 to be sent to the VAL II controller as described in the STOP_ALTER function.

DEFINE_FIXED_FRAME

As described earlier, all motions are specified to the controller in the world coordinate frame of the robot. If we wish to specify motion in any other frame, then we must supply the transformation data relating the new frame to the world coordinate frame. The application program passes this transformation data through the array FRAME_BUFF in the struct MOTN_RQST to the function DEFINE_FIXED_FRAME which stores it in the buffer FIX_FRAME. Only the rotation portion of the matrix is required, scaled by 16384, as int_16. The order of the matrix elements is as follows:

$$\begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$$

Since ALTER communication must be maintained on a 28 ms cycle, this function sends a "no motion" command to the controller as it requires almost a full cycle to execute.

MOVE

In Alt_server() all motions are executed using the function Transrot(). Although actual motions sent to the controller are given in incremental world mode (ALTER mode 23), Transrot() additionally accommodates motions in dynamic tool or fixed frames as described earlier. The frame in which the motions are to be specified is provided by the FRAME variable of the struct MOTN_RQST.

Both the dynamic tool frame and the fixed frame cases use the function Tform() to provide the matrix multiplication to translate the data into world coordinates. The tool frame motions take the conversion data from the array CUR_FRAME, which is updated every cycle, over the ALTER link with

current frame data from the Val II controller. The fixed frame conversion uses the values in the array FIX_FRAME which have been stored there by the DEFINE_FIXED_FRAME function before the motion sequence has begun.

The routine Vavpint is a part of the standard Harmony release. _Out_packet and _In_packet listings are included in Appendix A. These routines satisfy the communication with the serial ALTER link. Vavpint receives and transmits complete message packets over ALTER and among other housekeeping must pack and unpack the data according to the protocol detailed in the VAL II manual part 3 (3). _In_packet and _Out_packet are the interface between Vavpint and the higher level software just described.

ANGLE DEFINITION

In this section, we will define various methods for describing the orientation of the end effector of a manipulator. We will also work out the transformations between the different representations. The reference for this section is Paul (6).

Methods for Describing Position and Orientation

Specification of Position and Orientation

The position and orientation (pose) of a manipulator can be described by the pose of a coordinate frame attached to its last link. This coordinate frame is described by the homogenous transformation

$$[1] \qquad \mathbf{T} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{R} & \underline{p} \\ \hline \underline{0} & 1 \end{array}\right]$$

where the vector $\underline{p} = [p_x\ p_y\ p_z]^T$ specifies the position of the coordinate frame, and the unit vectors $\underline{n}$, $\underline{o}$, $\underline{a}$ lie along the coordinate frame x, y, and z axis, respectively, and describe its orientation (see Fig. 2). That is, the orientation is defined as the 3 x 3 matrix located on the top left of the **T** matrix and known as the **R** matrix. The vectors $\underline{n}$, $\underline{o}$, and $\underline{a}$ are known as the normal, the orientation, and the approach vectors. The values n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, and a_z are the components of the projections of the three unit vectors $\underline{n}$, $\underline{o}$, and $\underline{a}$ along the world frame x_w, y_w, and z_w axes, respectively.

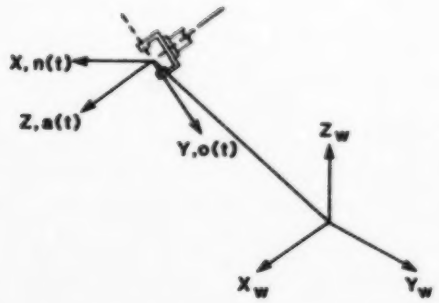

Figure 2. Position and orientation vectors at end effector

Orientation in Euler Angles

In the previous section, the orientation was defined as a function of the $\underline{n}$, $\underline{o}$, and $\underline{a}$ vectors. The orientation may also be defined in terms of Euler angles with respect to the world coordinates. Initially, let $\underline{n}(t_0)$, $\underline{o}(t_0)$, and $\underline{a}(t_0)$ align with x_w, y_w, and z_w, respectively, as shown in Fig. 3a. Any orientation $[\underline{n}(t), \underline{o}(t), \underline{a}(t)]$ may be obtained by a rotation of ϕ_e about z_w so that $o(t_0)$ aligns with $o(t_i)$ (subscript i indicates intermediate values), then a rotation of θ_e about $o(t_i)$, the new y, so that $a(t_0) = a(t_i)$ aligns with a(t), and finally a rotation of ψ_e about a(t), the new z, to obtain the required n(t) and o(t). This is equivalent to rotating the $[\underline{n}(t_0), \underline{o}(t_0), \underline{a}(t_0)]$ coordinate, which aligns with $[x_w, y_w, z_w]$ originally, ψ_e about z_w, then θ_e about y_w, and finally ϕ_e about z_w. These three consecutive rotations can be represented by

$$[2]\quad R_{Euler} = R(\phi_e, \theta_e, \psi_e) = \mathrm{Rot}(z, \phi_e)\,\mathrm{Rot}(y, \theta_e)\,\mathrm{Rot}(z, \psi_e)$$

$$= \begin{bmatrix} \cos\phi_e & -\sin\phi_e & 0 \\ \sin\phi_e & \cos\phi_e & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_e & 0 & \sin\theta_e \\ 0 & 1 & 0 \\ -\sin\theta_e & 0 & \cos\theta_e \end{bmatrix} \begin{bmatrix} \cos\psi_e & -\sin\psi_e & 0 \\ \sin\psi_e & \cos\psi_e & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\phi_e\, c\theta_e\, c\psi_e - s\phi_e\, s\psi_e & -c\phi_e\, c\theta_e\, s\psi_e - s\phi_e\, c\psi_e & c\phi_e\, s\theta_e \\ s\phi_e\, c\theta_e\, c\psi_e + c\phi_e\, s\psi_e & -s\phi_e\, c\theta_e\, s\psi_e + c\phi_e\, c\psi_e & s\phi_e\, s\theta_e \\ -s\theta_e\, c\psi & s\theta_e\, s\psi_e & c\theta_e \end{bmatrix}$$

where $c\phi_e = \cos\phi_e$ and $s\phi_e = \sin\phi_e$.

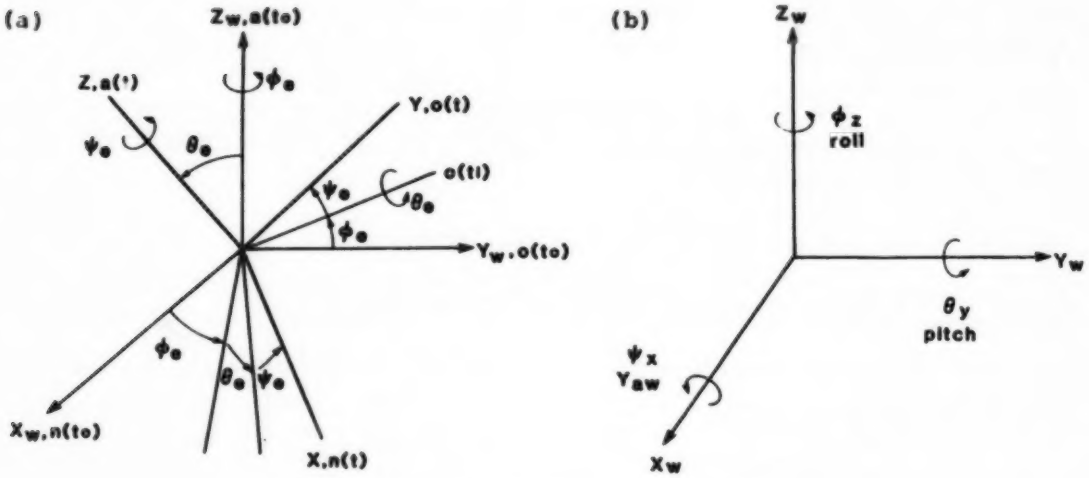

Figure 3(a) Euler angles; (b) RPY angles.

The PUMA Orientation Angles

The orientation of the PUMA robot arm is represented by the angles [O, A, T] which are basically Euler angles with an offset. The interpretation of the [O, A, T] angles is as follows, where the subscripts w and t stand for world and tool, respectively.

O angle is the angle between $-y_w$ and the z_t projection on the x_w-y_w plane. It is equivalent to the Euler angle $\phi_e + 90°$. The value of the angle O changes with the rotation of the projection of z_t around z_w.

A angle is the angle between z_w and z_t minus 90°. It is equivalent to the Euler angle $\theta_e - 90°$. The angle A represents the slope of the approach vector with respect to the horizontal plane x_w-y_w.

T angle is the angle between the projection of z_w on the x_t-y_t plane and the $-x_t$ axis. It is equivalent to the Euler angle ψ_e. The value of the angle T changes with the rotation of the plane x_t-y_t around z_t.

Orientation in Roll, Pitch, and Yaw

Roll, pitch, and yaw (RPY) are frequently used rotations. They are useful when we deal with rotations in the tool coordinates and also when using the ALTER mode. Roll corresponds to a rotation ϕ_z about the z_w, pitch

corresponds to a rotation θ_y about the y_w, and yaw corresponds to a rotation ψ_x about the x_w (see Fig. 3b).

For a rotation of ψ_x about station x-axis, followed by a rotation θ_y about station y-axis, and finally, a rotation ϕ_z about station z-axis, the order of rotation is

$$[3]\quad R_{RPY} = R(\phi_z, \theta_y, \psi_x) = \mathrm{Rot}(z, \phi_z)\,\mathrm{Rot}(y, \theta_y)\,\mathrm{Rot}(x, \psi_x)$$

$$= \begin{bmatrix} \cos\phi_z & -\sin\phi_z & 0 \\ \sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi_x & -\sin\psi_x \\ 0 & \sin\psi_x & \cos\psi_x \end{bmatrix}$$

$$= \begin{bmatrix} c\phi_z\, c\theta_y & c\phi_z\, s\theta_y\, s\psi_x - s\phi_z\, c\psi_x & c\theta_z\, s\theta_y\, c\psi_x + s\phi_z\, s\psi_x \\ s\phi_z\, c\theta_y & s\phi_z\, s\theta_y\, s\psi_x + c\phi_z\, c\psi_x & s\phi_z\, s\theta_y\, c\psi_x - c\phi_z\, s\psi_x \\ -s\theta_y & c\theta_y\, s\psi_x & c\theta_y\, c\psi_x \end{bmatrix}$$

where $c\phi_z = \cos\phi_z$ and $s\phi_z = \sin\phi_z$.

Relationship Between the Different Representations

The transformations between the Euler and RPY angles and the matrix R representing the $\underline{n}$, $\underline{o}$, and $\underline{a}$ vectors are presented in both directions. Also derived is the transformation between the RPY angles and the Euler (O, A, T) angles.

Transformations between R and the Euler/RPY Angles

Case 1:

Given the Euler angles ϕ_e, θ_e, ψ_e (or equivalently the O, A, T angles) or the RPY angles ϕ_z, θ_y, ψ_x, find the elements of the R matrix.

By straightforward substitution in equation [2] for the Euler angles or in equation [3] for the RPY angles, the vectors $\underline{n}$, $\underline{o}$, and $\underline{a}$ are found.

Case 2:

Given the vectors, $\underline{n}$, $\underline{o}$, and $\underline{a}$ find the Euler and RPY angles. The solution exists in Chapter 3, ref. 6, as follows:

(a) Euler transform solution (equations 3-40 to 3-43)

$$\phi_e = \text{atan2}\ (a_y,\ a_x)$$

and

$$\phi_e = \phi_e + 180°$$

$$\theta_e = \text{atan2}\ (\cos\ \phi_e\ a_x + \sin\ \phi_e\ a_y,\ a_z)$$

$$\psi_e = \text{atan2}\ (-\sin\ \phi_e\ n_x + \cos\ \phi_e\ n_y,\ -\sin\ \phi_e\ o_x + \cos\ \phi_e\ o_y)$$

A degeneracy occurs if $a_x = a_y = 0$ and in this case ϕ_e is arbitrarily set to zero.

(b) RPY transform solution (equations 3.50, 3.51, 3.54, 3.57)

$$\phi_z = \text{atan2}\ (n_y,\ n_x)$$

and

$$\phi_z = \phi_z + 180°$$

$$\theta_y = \text{atan2}\ (-n_z,\ \cos\ \phi_z\ n_x + \sin\ \phi_z\ n_y)$$

$$\psi_x = \text{atan2}\ (\sin\ \phi_z\ a_x - \cos\ \phi_z\ a_y,\ -\sin\ \phi_z\ o_x + \cos\ \phi_z\ o_y)$$

Transformation Between RPY Angles and Euler Angles

Given the RPY angles ϕ_z, θ_y, and ψ_x, find the Euler angles ϕ_e, θ_e, and ψ_e. By equating equivalent elements in R_{Euler} and R_{RPY}, the following solution is obtained:

$$\left.\frac{o_z}{n_z}\right|_{Euler} = \left.\frac{o_z}{n_z}\right|_{RPY}$$

$$\frac{\sin\ \theta_e\ \sin\ \psi_e}{-\sin\ \theta_e\ \cos\ \psi_e} = \frac{\cos\ \theta_y\ \sin\ \psi_x}{-\sin\ \theta_y}$$

or

[4] $$\psi_e = \text{atan2}\ (\sin\ \psi_x,\ \tan\ \theta_y)$$

ψ_e is equal to the PUMA T angle and represents in Euler definition the second rotation about the z-axis.

$$\left.\frac{o_z}{a_z}\right|_{Euler} = \left.\frac{o_z}{a_z}\right|_{RPY}$$

$$\frac{\sin\theta_e \sin\psi_e}{\cos\theta_e} = \frac{\cos\theta_y \sin\psi_x}{\cos\theta_y \cos\psi_x}$$

[5] $\theta_e = \text{atan2}(\tan\psi_x, \sin\psi_e)$

θ_e is equal to the PUMA A angle + $\pi/2$ and represents, in Euler, the rotation about the y-axis.

$$\left.\frac{a_y}{a_x}\right|_{Euler} = \left.\frac{a_y}{a_x}\right|_{RPY}$$

[6] $\phi_e = \text{atan2}(\sin\phi_z \sin\theta_y \cos\psi_x - \cos\phi_z \sin\psi_x, \cos\theta_z \sin\theta_y \cos\psi_x + \sin\phi_z \sin\psi_x)$

ϕ_e is equal to the PUMA O angle - $\pi/2$ and represents the first rotation around the z-axis.

Special Cases:

(a) $\theta_y = 0$, $\phi_z = 0$, that is, pure rotation ψ_x about x. Solving equations [4]-[6], we obtain

$$\theta_e = \psi_x$$

$$\psi_e = \text{atan2}(\sin\psi_x, 0)$$

$$\phi_e = \text{atan2}(-\sin\psi_x, 0)$$

if $\sin\psi_x$ is positive, then

$$\psi_e = 90°$$

$$\phi_e = -90°$$

but if $\sin\psi_x$ is negative

$$\psi_e = -90°$$

$$\phi_e = 90°$$

and consequently

$$O = 0 \text{ or } 180°$$

$$A = \psi_x - 90°$$

$$T = 90° \text{ or } -90°$$

(b) $\psi_x = 0$, $\phi_z = 0$, that is, a pure rotation θ_y about y, and by the definition of Euler angles

$$\theta_e = \theta_y$$

$$\psi_e = \text{atan2}\ (0, \tan \theta_y) = 0$$

$$\phi_e = \text{atan2}\ (0, \sin \theta_y) = 0$$

and consequently

$$O = 90°$$

$$A = \theta_y - 90°$$

$$T = 0$$

(c) $\psi_x = 0$, $\theta_y = 0$, that is, a pure rotation ϕ_z about z and by the definition of Euler angles

$$\theta_e = 0$$

It follows that

$$\phi_e + \psi_e = \phi_z$$

Assuming

$$\psi_e = 0,$$

then

$$\phi_e = \phi_z$$

and consequently

$$O = \phi_z + 90°$$

$$A = -90°$$

$$T = 0$$

Example:

Given the pose **A** of an end effector and a transformation **C**, the goal pose **B** of the end effector is found by postmultiplying the frame **A** by the transformation **C** (known in the Unimation VAL as compound transformation)

$$B = A:C$$

Assuming that the transformation **C** is a pure rotation defined by the following RPY angles

$$\psi_x = 20°, \quad \theta_y = 30°, \quad \phi_z = 25°$$

the equivalent OAT angles can be found using equations [4] to [6] as follows

$$T = \psi_e = 30.6423°$$

$$A = \theta_e - 90° = -54.4686°$$

$$O = \phi_e + 90 = 78.9486°$$

and consequently the transformation **C** is

$$C = [0, \quad 0, \quad 0, \quad 78.9486, \quad -54.4686, \quad 30.6423]^T$$

An alternative way to reach the goal pose **B** is to compute three transformations each corresponding to a rotation about a single axis. In other words, the first transformation (C_1) would cause a rotation of 20° about the stationary x-axis, the second (C_2) a rotation of 30° about the stationary y-axis, and finally the third (C_3) a rotation of 25° about the stationary z-axis. This is accomplished by using the special cases a, b, and c, and then using the compound transformation in the order shown to find **B**.

$$B = A:C_3:C_2:C_1$$

where

$$C_1 = [0, \quad 0, \quad 0, \quad 0, \quad -70, \quad 90]^T \qquad \text{case (a)}$$

$$C_2 = [0, \quad 0, \quad 0, \quad 90, \quad -60, \quad 0]^T \qquad \text{case (b)}$$

$$C_3 = [0, \quad 0, \quad 0, \quad 115, \quad -90, \quad 0]^T \qquad \text{case (c)}$$

SUMMARY

This report has provided a detailed description of the ALTER implementation for total manipulator control from a Chorus/Harmony computer. An application program can use these low level control routines to provide flexible manipulator control in Cartesian space. This report also includes a unified description of the orientation representations required for PUMA control.

ACKNOWLEDGEMENT

The authors acknowledge the support of David Green and W. Morven Gentleman in installing this facility on the Chorus/Harmony system.

REFERENCES

1. W.M. Gentleman. Using the Harmony Operating System. NRC/ERB-966. National Research Council of Canada, Ottawa, Ont. December 1983.

2. D.A. Green. Chorus - A multiprocessor architecture for real-time control applications. NRC/ERB-964. National Research Council of Canada, Ottawa, Ont. December 1983.

3. Programming manual user's guide to Val II 398T1 version 1.1. Unimation Inc., Shelter Rock Lane, Danbury, CT. August 1984

4. S.E. Elgazzar, D. Green, and D. O'Hara. A vision-based robot system using a multiprocessor controller. NRC/ERB-971. National Research Council of Canada, Ottawa, Ont. June 1984

5. D.H. O'Hara and R. Kurtz. Prismatic shaped block insertion: an application of a multiprocessor robot controller using Harmony. NRC/ERB-1006, National Research Council of Canada, Ottawa, Ont. In press.

6. R.P. Paul. Robot manipulators: mathematics, programming and control. The MIT Press 1981.

APPENDIX A

The code listed here has been edited. Its function is to aid in understanding routines only.

Program DEMOALT; This program runs in the VAL II controller.

```
PROGAM  DEMOALT

        DT=.1E-01
        MODE=23

        ALTER ( 0, MODE )          * Start ALTER
        PCEXECUTE RDCODE,-1,0      * Monitor the Exception Code

        DO

           DELAY DT                * Controlled Path Modification

        UNTIL STCOD == 127         * Stop ALTER
        NOALTER                    * Disconnect ALTER

        PCEND

END


PROGRAM RDCODE

STPCOD = EXCEPTION                 * Get the EXCEPTION CODE
WAIT                               * Synchronized

END
```

This code runs in Chorus

```
typedef struct STD_RQST
  {
    uint_16             MSG_SIZE;
    int_16              MSG_TYPE;
  };

typedef struct STD_RPLY
  {
    uint_16             MSG_SIZE;
    uint_32             RESULT;
  };


typedef struct VALI_MSG
{
    uint_16             MSG_SIZE;
    uchar               MSG_BUFF[64];
};


typedef struct VALO_MSG
{
    uint_16             MSG_SIZE;
    uint_16             MSG_TYPE;
    uchar               MSG_BUFF[14];
};


typedef struct MOTN_RQST
  {
    struct STD_RQST             STD_REQUEST;
    int_16                      EXECUTE;
    int_16                      MODE;
    char                        FRAME;
    int_16                      REPETITIONS;
    char                        EXCEPTION;
    int_16                      FRAME_BUFF[12];
    float                       MOTION_BUFF[6];
  };
```

```
typedef struct ALT_RQST

  {
    struct STD_RQST              STD_REQUEST;
    int_16                       ERROR;
    int_16                       SETPOINT_BUFF[12];
  };


typedef struct MOTN_RPLY
  {
    struct MOTN_RQST             MOTN_REQUEST;
    int_16                       ERROR;
    int_16                       SETPOINT_BUFF[12];
  };
```

```
Alt_server()
{

  extern struct VALI_MSG Cur_Frame, Fix_Frame;

  char alter_up;
  struct MOTN_RQST mot_rqst;
  struct ALT_RQST mot_rply;
  int_16 i, tmp;


  Init_alt_serv();

  alter_up = FALSE;


  for(;;)
  {
     mot_rqst.STD_REQUEST.MSG_SIZE = sizeof( mot_rqst );
     _Receive( (char *)&mot_rqst, _Father_id() );

     switch ( mot_rqst.MODE )
     {

       case IDLE:

         if ( alter_up )
         {
           Idle( mot_rqst.EXCEPTION );
           mot_rply.ERROR = ALTER_OK;
         }
         else mot_rply.ERROR = ALTER_DOWN;
         break;

       case MOVE:

         if ( alter_up )
         {
           Transrot( mot_rqst.FRAME,
                     mot_rqst.EXCEPTION,
                     mot_rqst.MOTION_BUFF[0],
                     mot_rqst.MOTION_BUFF[1],
                     mot_rqst.MOTION_BUFF[2],
                     mot_rqst.MOTION_BUFF[3],
                     mot_rqst.MOTION_BUFF[4],
```

```
                mot_rqst.MOTION_BUFF[5]
              );
    mot_rply.ERROR = ALTER_OK;
  }
  else mot_rply.ERROR = ALTER_DOWN;
  break;

case DEFINE_FIXED_FRAME:

  if ( alter_up )
  {
    Idle( mot_rqst.EXCEPTION );
    for( i=0; i < 9; i++ )
    {
      Fix_Frame.MSG_BUFF[2*i+6] = mot_rqst.FRAME_BUFF[i] & 0x00ff;
      Fix_Frame.MSG_BUFF[2*i+7] = mot_rqst.FRAME_BUFF[i] >> 8;

    }
    mot_rply.ERROR = ALTER_OK;
  }
  else mot_rply.ERROR = ALTER_DOWN;
  break;

case STOP_ALTER:

  if ( alter_up )
  {
    Stp_Alter();
    alter_up = FALSE;
    mot_rply.ERROR = ALTER_OK;
  }
  else mot_rply.ERROR = ALTER_DOWN;
  break;

case START_ALTER:

  if ( alter_up ) mot_rply.ERROR = ALTER_UP;
  else
  {
    mot_rply.ERROR = ALTER_OK;
    Ini_Alter();  /* Must be synchronized with the valpak command */
                  /* execute demoalt.
    alter_up = TRUE;
  }
  break;
```

```
        default:

          _Breakpoint( "Alt_server : MODE error" );
          break;

      }

      for ( i=0; i < 12; i++ )
      {
        tmp = Cur_Frame.MSG_BUFF[2*i+7];
        mot_rply.SETPOINT_BUFF[i] =
           ( tmp << 8 ) | Cur_Frame.MSG_BUFF[2*i+6];
      }

      mot_rply.STD_REQUEST.MSG_TYPE = ALTER_RQST;
      mot_rply.STD_REQUEST.MSG_SIZE = sizeof( struct ALT_RQST );
      _Reply( (char *)&mot_rply, _Father_id() );

  }

};
```

```
Init_alt_serv()
{


  char  *uart;
  uint_32 *nmi;


/* Initialize */
  nmi = 0x7c;
  *nmi = Vavpint;                    /* stuff level 7 int vector */
  uart = 0xFFFF21;
  *uart = RESET;
  *uart = SETUP;


  Outpakid= _Create(_OUT_PACKET);
  Inpakid = _Create(_IN_PACKET);

};
```

```
Ini_Alter()


{
 extern uint_32 Inpakid,Outpakid;
 extern uchar  *Outpacke;
 char ackcod;                        /* ackcod could be 0,1 or -1 */
 struct STD_RQST inp_rqst;
 struct VALI_MSG From_Val;
 struct STD_RQST out_rqst;
 struct VALO_MSG To_Val;

 Outpacke = 0;

 /* Receive */
       From_Val.MSG_SIZE= sizeof(struct VALI_MSG) ;
      _Receive( (char *)&From_Val , Inpakid ); /* Get NormlMsg */
       inp_rqst.MSG_SIZE= sizeof(struct STD_RQST) ;
      _Reply( (char *)&inp_rqst , Inpakid );   /* Get the inpak going */

 /* Check */
       if( (From_Val.MSG_BUFF[0] & 7 ))
       {
        To_Val.MSG_SIZE = 5;
        To_Val.MSG_TYPE = 0;
        To_Val.MSG_BUFF[0] = 1;   /* NOK */
        out_rqst.MSG_SIZE = sizeof(struct STD_RQST);
        _Receive( (char *)&out_rqst , Outpakid );
        _Reply( (char *)&To_Val , Outpakid );
        return(1);
       }
       else
       {
        To_Val.MSG_SIZE = 5;
        To_Val.MSG_TYPE = 0;
        To_Val.MSG_BUFF[0] = 0;  /* OK */

        out_rqst.MSG_SIZE = sizeof(struct STD_RQST);
        _Receive( (char *)&out_rqst , Outpakid );
 /* Send */
        _Reply( (char *)&To_Val , Outpakid );
        return(0);
       }

}
```

```
Idle ( ec )

char ec;
{
 extern uint_32 Inpakid,Outpakid;
 extern struct VALI_MSG Cur_Frame;

 struct STD_RQST inp_rqst;
 struct STD_RQST out_rqst;
 struct VALO_MSG To_Val;

 /* Receive */
       Cur_Frame.MSG_SIZE= sizeof(struct VALI_MSG) ;
      _Receive( (char *)&Cur_Frame , Inpakid ); /* Get NormalMsg */
       inp_rqst.MSG_SIZE= sizeof(struct STD_RQST) ;
      _Reply( (char *)&inp_rqst , Inpakid );   /* Get the inpak going */

 /* Check */
       if( (Cur_Frame.MSG_BUFF[0] & 7 ))
       {
       To_Val.MSG_SIZE = 6;
       To_Val.MSG_TYPE = 1;
       To_Val.MSG_BUFF[0] = 0;          /* Exception code */
       To_Val.MSG_BUFF[1] = 0;          /* Nomotion */
       out_rqst.MSG_SIZE = sizeof(struct STD_RQST);
       _Receive( (char *)&out_rqst , Outpakid );
       _Reply( (char *)&To_Val , Outpakid );
       return(1);
       }
       else
       {
       To_Val.MSG_SIZE = 6;
       To_Val.MSG_TYPE = 1;
       To_Val.MSG_BUFF[0] = ec;          /* Exception code */
       To_Val.MSG_BUFF[1] = 0;          /* Nomotion */

       out_rqst.MSG_SIZE = sizeof(struct STD_RQST);
       _Receive( (char *)&out_rqst , Outpakid );
 /* Send */
       _Reply( (char *)&To_Val , Outpakid );
       return(0);
       }
}
```

```
Transrot(mode,ec,dx,dy,dz,rx,ry,rz)

char ec,mode;
float dx,dy,dz,rx,ry,rz;


{
 extern uint_32 Inpakid,Outpakid;
 extern struct VALI_MSG Fix_Frame, Cur_Frame;

 int_16 tmpd, i;
 char w_high, w_low;
 struct STD_RQST inp_rqst;
 struct STD_RQST out_rqst;
 struct VALO_MSG To_Val;
 int_32 D[6], TR[6];

 /* Receive */
       Cur_Frame.MSG_SIZE= sizeof(struct VALI_MSG) ;
      _Receive( (char *)&Cur_Frame , Inpakid ); /* Get NormlMsg */
       inp_rqst.MSG_SIZE= sizeof(struct STD_RQST) ;
      _Reply( (char *)&inp_rqst , Inpakid );   /* Get the inpak going */

 /* Check */
       if( (Cur_Frame.MSG_BUFF[0] & 7 ))
       {
        To_Val.MSG_SIZE = 6;
        To_Val.MSG_TYPE =  1;
        To_Val.MSG_BUFF[0] = 0;          /* Exception code */
        To_Val.MSG_BUFF[1] = 0;          /* Idle  */
       _Receive( (char *)&out_rqst , Outpakid );
       _Reply( (char *)&To_Val , Outpakid );
        return(1);
       }
       else
       {
        To_Val.MSG_SIZE = 18;
        To_Val.MSG_TYPE =  1;
        To_Val.MSG_BUFF[0] = ec;          /* Exception code */
        To_Val.MSG_BUFF[1] = 0x3F;        /* Modify dx,dy,dz,rx,ry,rz */
```

```
/*  Scale the data.   */

        if ( mode == WORLD )
        {
         TR[0] = dx * SCALEF1;
         TR[1] = dy * SCALEF1;
         TR[2] = dz * SCALEF1;
         TR[3] = rx * SCALEF2;
         TR[4] = ry * SCALEF2;
         TR[5] = rz * SCALEF2;
        }
        else
        {
         D[0] = dx * SCALEF1;
         D[1] = dy * SCALEF1;
         D[2] = dz * SCALEF1;
         D[3] = rx * SCALEF2;
         D[4] = ry * SCALEF2;
         D[5] = rz * SCALEF2;
         if ( mode == FIXED ) Tform( &Fix_Frame,D,TR );

         else Tform( &Cur_Frame,D,TR );
        }

        for ( i=0; i < 6; i++ )
        {
         w_low = TR[i] & 0x00ff;
         w_high = TR[i] >> 8;
         To_Val.MSG_BUFF[2+2*i] = w_low;
         To_Val.MSG_BUFF[3+2*i] = w_high;
        }

        out_rqst.MSG_SIZE = sizeof(struct STD_RQST);
       _Receive( (char *)&out_rqst , Outpakid );
 /* Send */
       _Reply( (char *)&To_Val , Outpakid );
        return(0);
       }

}
```

```
Stp_Alter()

{
 extern uint_32 Inpakid,Outpakid;
 struct STD_RQST inp_rqst;
 struct VALI_MSG From_Val;
 struct STD_RQST out_rqst;
 struct VALO_MSG To_Val;

  do
     {

 /* Receive */
       From_Val.MSG_SIZE= sizeof(struct VALI_MSG) ;
      _Receive( (char *)&From_Val , Inpakid ); /* Get NormlMsg */
       inp_rqst.MSG_SIZE= sizeof(struct STD_RQST) ;
      _Reply( (char *)&inp_rqst , Inpakid );   /* Get the inpak going */
 /* Check */
       if( (From_Val.MSG_BUFF[0] & 7 ))
       {
        To_Val.MSG_SIZE = 6;
        To_Val.MSG_TYPE = 1;
        To_Val.MSG_BUFF[0] = 0;
        To_Val.MSG_BUFF[1] = 0;          /* Nomotion */
        out_rqst.MSG_SIZE = sizeof(struct STD_RQST);
       _Receive( (char *)&out_rqst , Outpakid );
       _Reply( (char *)&To_Val , Outpakid );
        return(1);
       }
       else
       {
        if(From_Val.MSG_BUFF[0] != STPMSG )
        {
          To_Val.MSG_SIZE = 6;
          To_Val.MSG_TYPE = 1;
          To_Val.MSG_BUFF[0] = STPCOD;     /* STOP-Exception code */
          To_Val.MSG_BUFF[1] = 0;          /* Nomotion */

          out_rqst.MSG_SIZE = sizeof(struct STD_RQST);
          _Receive( (char *)&out_rqst , Outpakid );
          /* Send */
          _Reply( (char *)&To_Val , Outpakid );
        }
        else
        {
```

```
            break;
          }

        }
      }  while ( From_Val.MSG_BUFF[0] != STPMSG );


      return(0);
}
```

```
Tform ( Val_Frame,ti,wi )

struct VALI_MSG *Val_Frame;
int_32 ti[], wi[];

{
   int_16 i,j;
   int_16 tmp;

/*
    Compute the matrix multiplications [dR] = [n|o|a] * [dRt] and [dT] =
    [n|o|a] * [dTt].
    The matrix [n|o|a] is found in Val_Frame, in the same format as alter
    specifies.
    As each entry in the transformation matrix [n|o|a] is scaled by 16384,
    the resulting world increments wi[i] are divided by 16384.
*/

   wi[0] = wi[1] = wi[2] = wi[3] = wi[4] = wi[5] = 0;

   for ( i=0; i<=2; i++ )
   {
      for ( j=0; j<=2; j++ )
      [
         tmp = Val_Frame->MSG_BUFF[2*i+6*j+7];
         tmp = ( tmp << 8 ) | Val_Frame->MSG_BUFF[2*i+6*j+6];
         wi[i] += tmp * ti[j];
         wi[i+3] += tmp * ti[j+3];
      }
      wi[i] = wi[i] >> 14;      /* Division by 16384 to correct for */
      wi[i+3] = wi[i+3] >> 14;  /* alter scaling.                   */
   }

};
```

```
_In_packet()
{
        struct STD_RQST inp_rqst;
        struct VALI_MSG inp_pckt;
        extern uint_32 in_state,in_begin;
        extern uchar  *Inpacket;
        for(;;)
        {

            Inpacket = &inp_pckt;
            in_state = in_begin;
            _Await_interrupt( INPACKET, 0 );      /* Get packet from VAL */
            inp_rqst.MSG_SIZE=sizeof(struct STD_RQST );
             _Send( (char *)&inp_pckt, (char *)&inp_rqst, _Father_id() );

         }

 };
```

```
_Out_packet()


{
   extern uint_32 out_stat,out_wait;
   extern uchar  *Outpacke;
   extern Start_pa();
   struct STD_RQST out_rqst;
   struct VALO_MSG out_pckt;


      for(;;)
      {

        out_stat = out_wait;
        out_rqst.MSG_SIZE= sizeof(struct STD_RQST);
        out_pckt.MSG_SIZE= sizeof(struct VALO_MSG);
        _Send( (char *)&out_rqst , (char *)&out_pckt ,_Father_id() );
         if( !(out_pckt.MSG_TYPE) )
         {
          out_pckt.MSG_TYPE = 3;
          Outpacke = &out_pckt.MSG_TYPE;
          Start_pa();
         _Await_interrupt( OUTPACKET, 0 );/* Packet has been send to VAL */

          out_stat = out_wait;
          out_pckt.MSG_TYPE = 4;
          out_pckt.MSG_BUFF[0] = 0;
          out_pckt.MSG_BUFF[1] = 0;
          Outpacke = &out_pckt.MSG_TYPE;

         }
         else
         {
          out_pckt.MSG_TYPE = out_pckt.MSG_SIZE - 2;
          Outpacke = &out_pckt.MSG_TYPE;
         }


        _Await_interrupt( OUTPACKET, 0 );/* Packet has been send to VAL */


      }

};
```